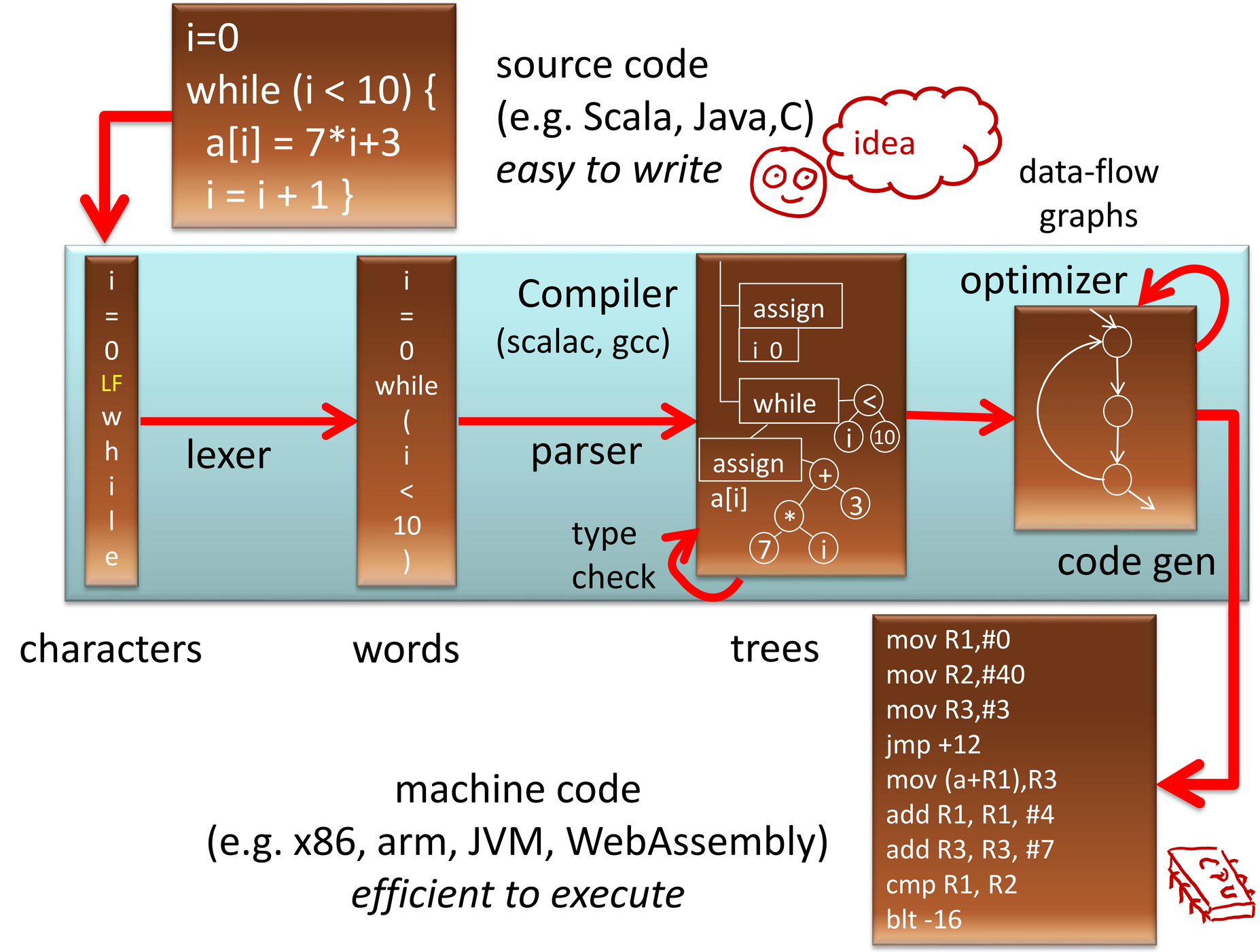
Code Generation: Introduction


i=0
while (i < 10) {
a[i] = 7*i+3
i = i + 1 }
source code
(e.g. Scala, Java,C)
easy to write
idea
data-flow
graphs
i
=
0
LF
w
h
i
l
e
lexer
i
=
0
while
(
i
<
10
)
Compiler
(scalac, gcc)
parser
assign
i 0
while
<
i
10
assign
a[i]
+
3
*
7
i
type
check
optimizer
code gen
characters
words
trees
mov R1,#0
mov R2,#40
mov R3,#3
jmp +12
mov (a+R1),R3
add R1, R1, #4
add R3, R3, #7
cmp R1, R2
blt -16
machine code
(e.g. x86, arm, JVM, WebAssembly)
efficient to execute

Example: gcc

test.c

```
#include <stdio.h>
int main() {
  int i = 0;
  int j = 0;
  while (i < 10) {
    printf("%d\n", j);
    i = i + 1;
    j = j + 2*i+1;
  }
}
```

gcc test.c -S

test.s

```
        jmp .L2
.L3:    movl -8(%ebp), %eax
        movl %eax, 4(%esp)
        movl $.LC0, (%esp)
        call printf
        addl $1, -12(%ebp)
        movl -12(%ebp), %eax
        addl %eax, %eax
        addl -8(%ebp), %eax
        addl $1, %eax
        movl %eax, -8(%ebp)
.L2:
        cmpl $9, -12(%ebp)
        jle .L3
```

What did (i<10) compile to?

javac example

```
while (i < 10) {
  System.out.println(j);
  i = i + 1;
  j = j + 2*i+1;
}
```

javac Test.java
javap –c Test

```
4: iload_1
5: bipush 10
7: if_icmpge 32
10: getstatic #2; //System.out
13: iload_2
14: invokevirtual #3; //println
17: iload_1
18: iconst_1
19: iadd
20: istore_1
21: iload_2
22: iconst_2
23: iload_1
24: imul
25: iadd
26: iconst_1
27: iadd
28: istore_2
29: goto 4
32: return
```

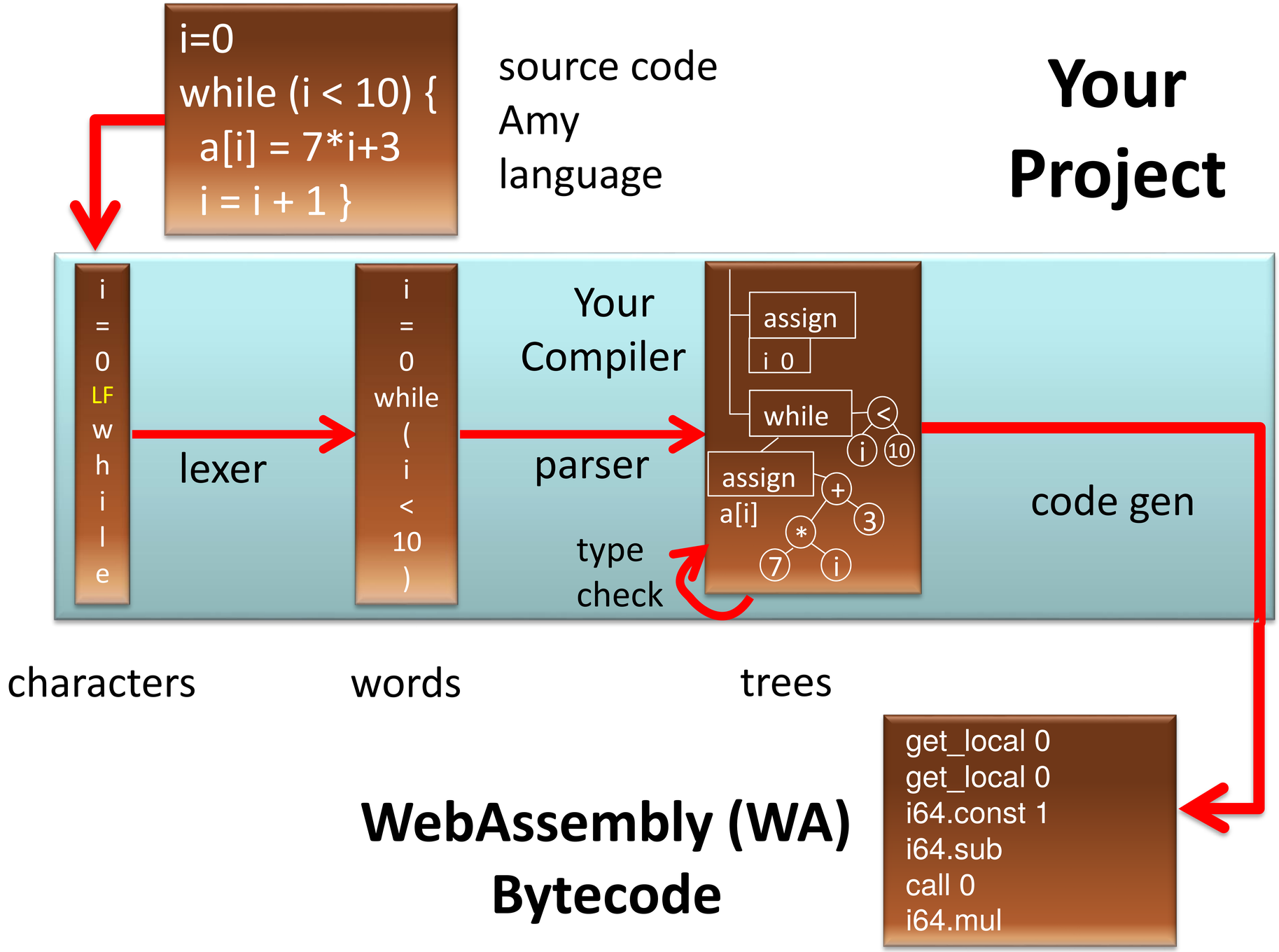
Guess what each JVM instruction for the highlighted expression does.

Java Virtual Machine

Use: **javac -g *.java** to compile

javap -c -l ClassName to explore

https://docs.oracle.com/javase/specs/jvms/se8/html/jvms-2.html#jvms-2.11


i=0
while (i < 10) {
a[i] = 7*i+3
i = i + 1 }
source code
Amy
language
Your
Project
i
=
0
LF
w
h
i
l
e
lexer
i
=
0
while
(
i
<
10
)
Your
Compiler
parser
assign
i 0
while
<
i
10
assign
a[i]
+
3
*
7
i
type
check
code gen
characters
words
trees
WebAssembly (WA)
Bytecode
get_local 0
get_local 0
i64.const 1
i64.sub
call 0
i64.mul

WebAssembly

- Overview of bytecodes: http://webassembly.org/docs/semantics/
- Compiling from C: http://webassembly.org/getting-started/developers-guide/ https://hacks.mozilla.org/2017/03/previewing-the-webassembly-explorer/
- Research paper and the talk:
  Bringing the Web up to Speed with WebAssembly
  by Andreas Haas, Andreas Rossberg, Derek Schuff, Ben L. Titzer, Dan Gohman, Luke Wagner, Alon Zakai, JF Bastien, Michael Holman.
  ACM SIGPLAN Conf. Programming Language Design and Implementation (PLDI), 2017.

WebAssembly example

C++

```
int factorial(int n) {
  if (n == 0)
    return 1;
  else
    return n * factorial(n-1);
}
```

WebAssembly

```
get_local 0         // n
i64.const 0         // 0
i64.eq              // n==0 ?
if i64
    i64.const 1     // 1
else
    get_local 0     //  n
    get_local 0     //  n
    i64.const 1     //  1
    i64.sub         // n-1
    call 0          // f(n-1)
    i64.mul         // n*f(n-1)
end
```

More at: https://mbebenita.github.io/WasmExplorer/

Stack Machine: High-Level Machine Code

stack

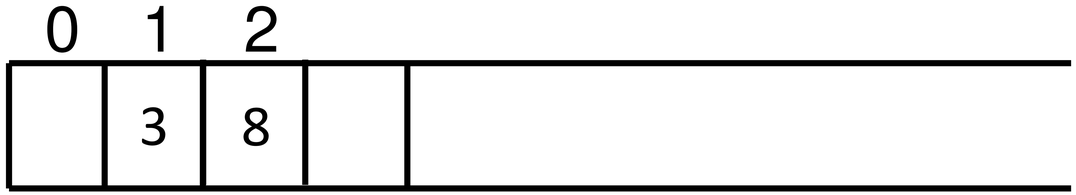
top of stack

instruction sequence:

program counter

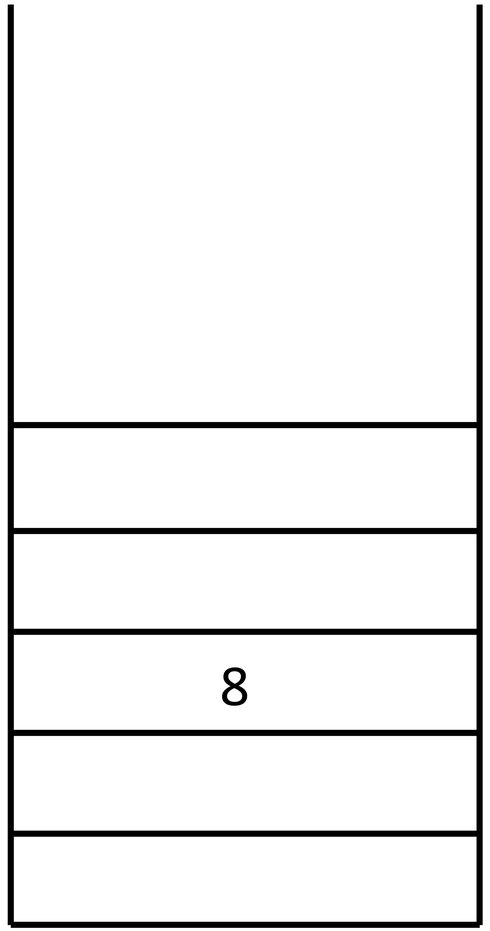
```
get_local 2
i64.const 2
get_local  1
i64.mul
i64.add
i64.const 1
i64.add
set_local 2
```

Memory (for locals):



Let us step through

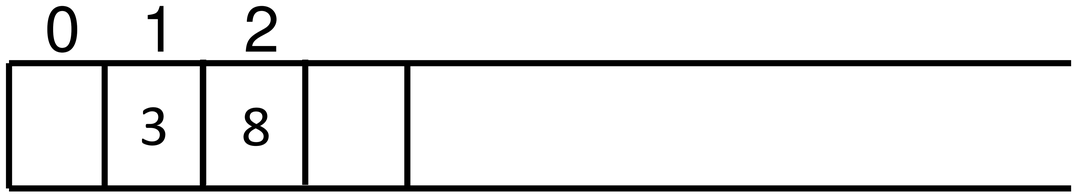
Operands are consumed from stack and put back onto stack



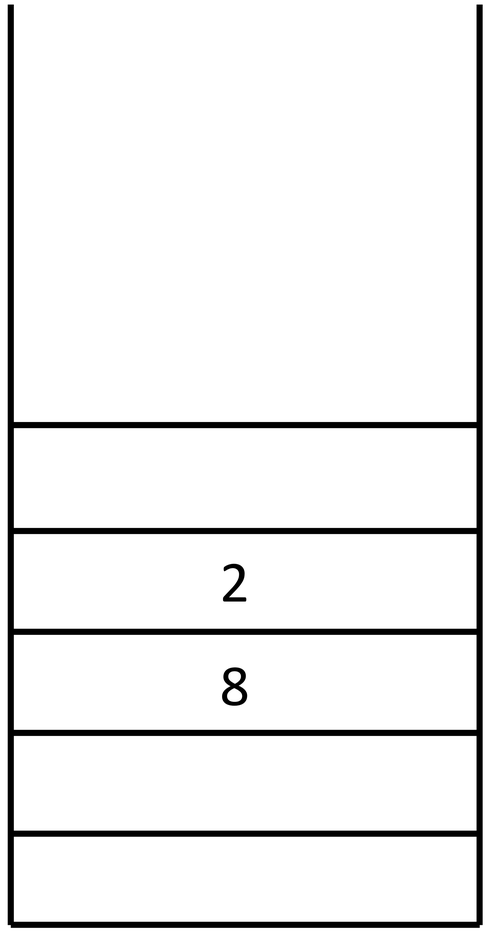
instruction sequence:

```
get_local 2
i64.const 2
get_local  1
i64.mul
i64.add
i64.const 1
i64.add
set_local 2
```

memory:



Operands are consumed from stack and put back onto stack



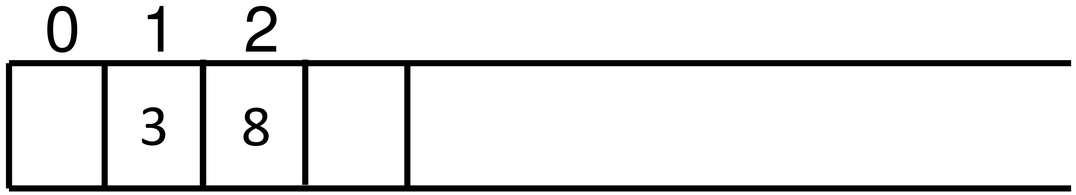
top of stack

stack

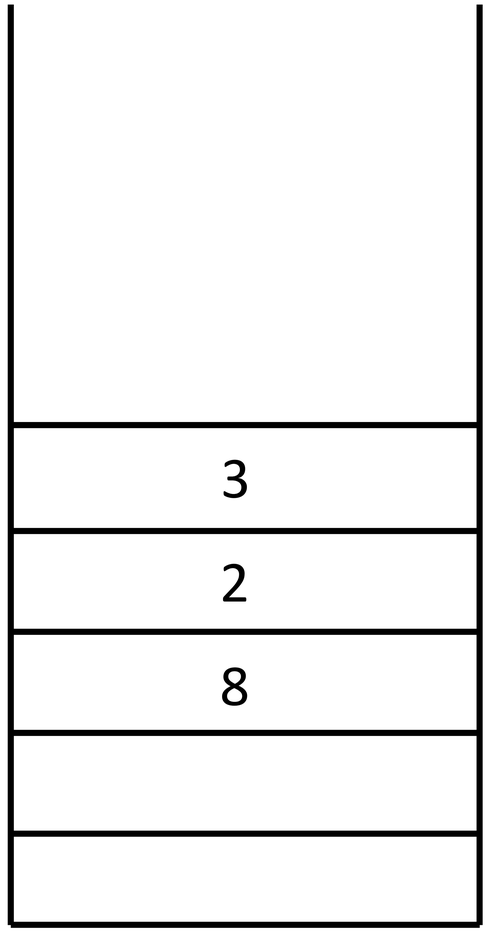
instruction sequence:

```
get_local 2
i64.const 2
get_local  1
i64.mul
i64.add
i64.const 1
i64.add
set_local 2
```

memory:



Operands are consumed from stack and put back onto stack

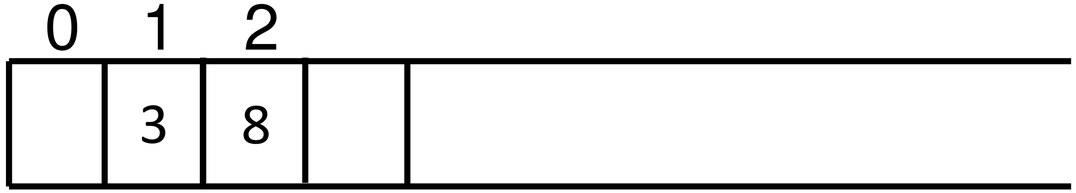


top of stack

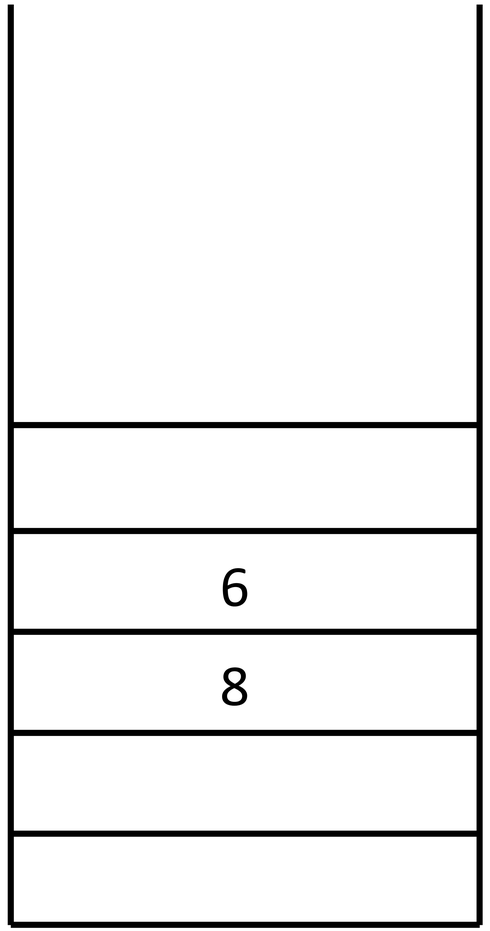
instruction sequence:

```
get_local 2
i64.const 2
get_local  1
i64.mul
i64.add
i64.const 1
i64.add
set_local 2
```

memory:



Operands are consumed from stack and put back onto stack



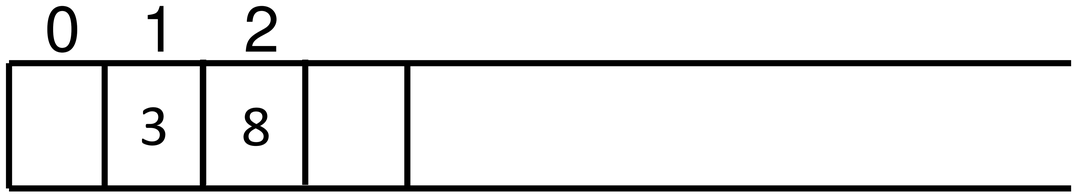
top of stack

stack

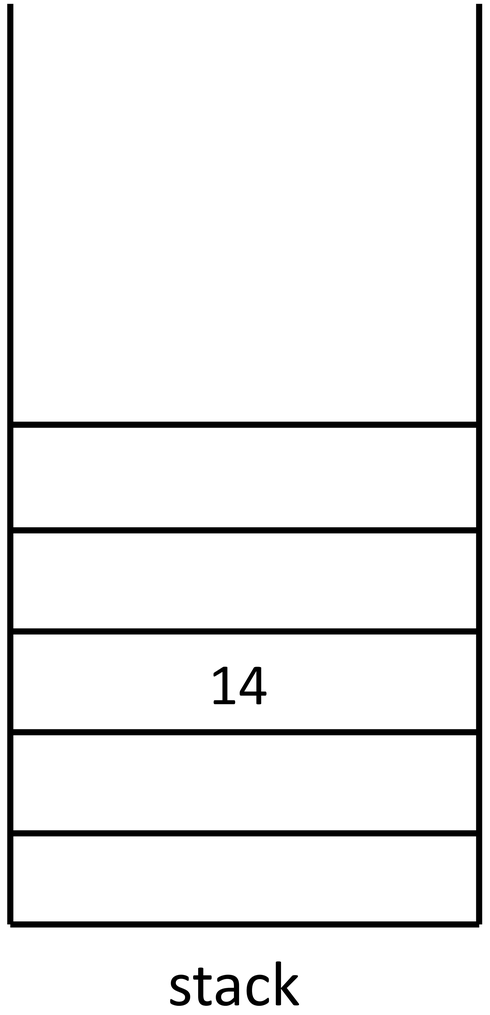
instruction sequence:

```
get_local 2
i64.const 2
get_local  1
i64.mul
i64.add
i64.const 1
i64.add
set_local 2
```

memory:



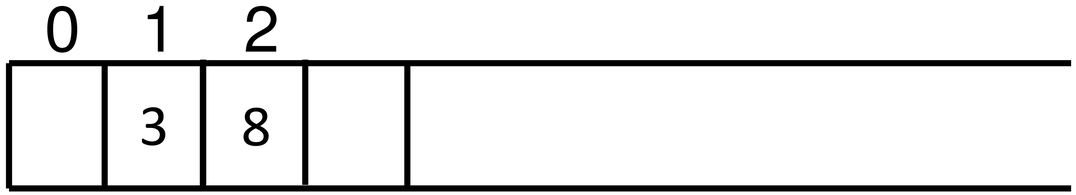
Operands are consumed from stack and put back onto stack



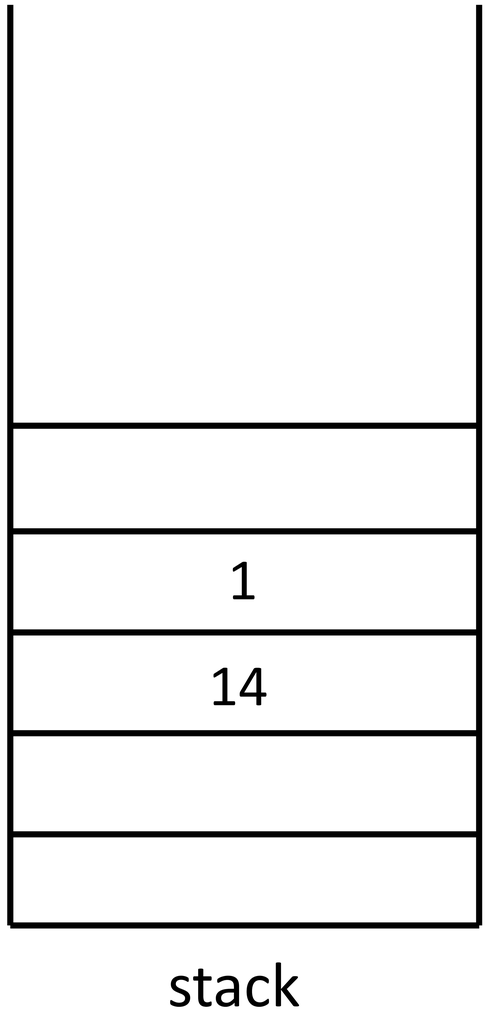
instruction sequence:

```
get_local 2
i64.const 2
get_local  1
i64.mul
i64.add
i64.const 1
i64.add
set_local 2
```

memory:



Operands are consumed from stack and put back onto stack

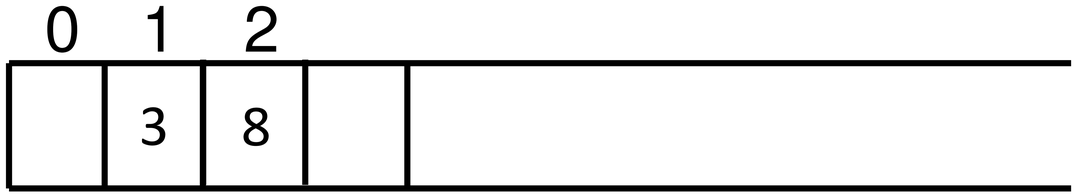


top of stack

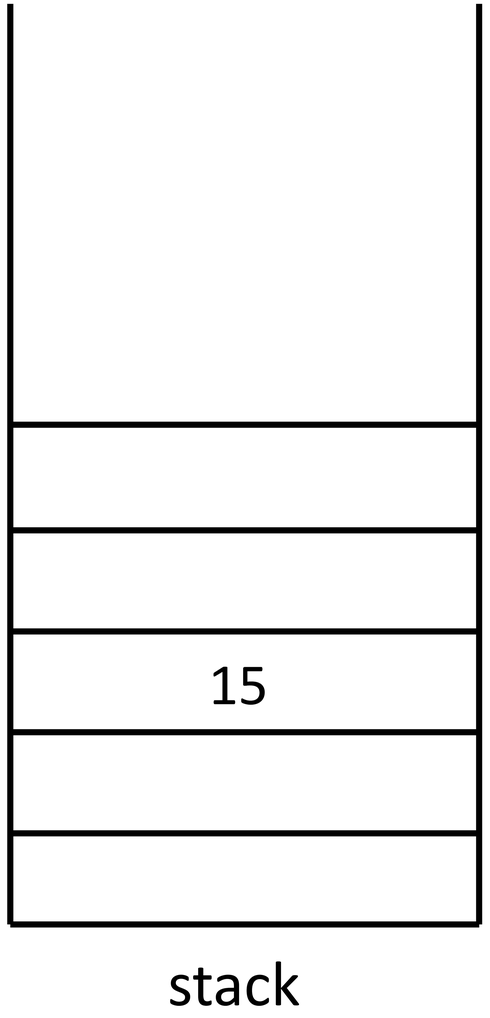
instruction sequence:

```
get_local 2
i64.const 2
get_local  1
i64.mul
i64.add
i64.const 1
i64.add
set_local 2
```

memory:



Operands are consumed from stack and put back onto stack

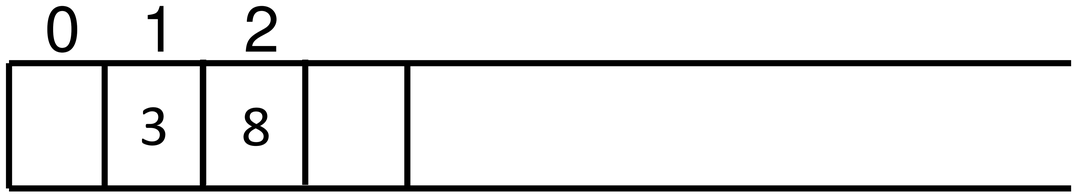


top of stack

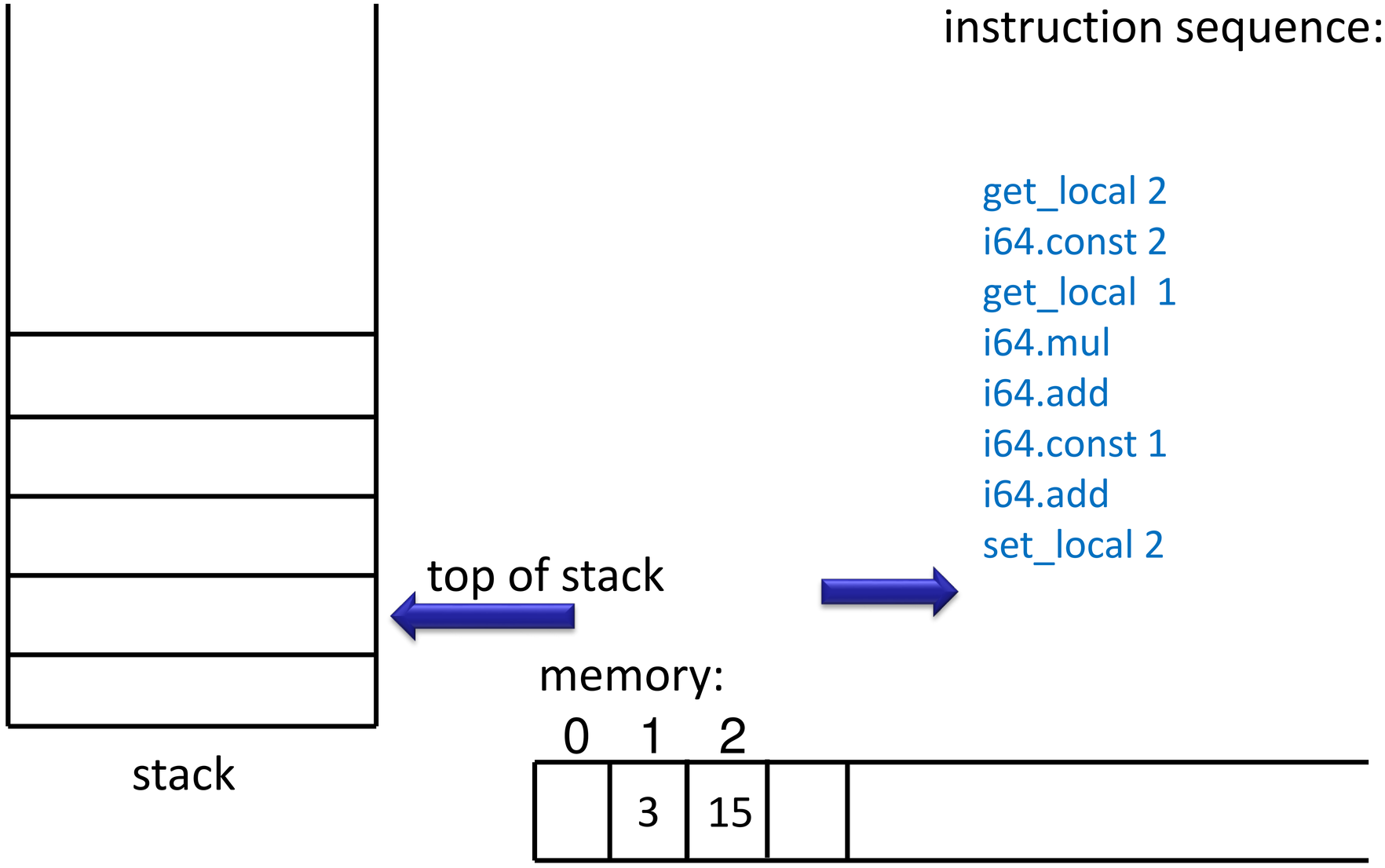
instruction sequence:

```
get_local 2
i64.const 2
get_local  1
i64.mul
i64.add
i64.const 1
i64.add
set_local 2
```

memory:

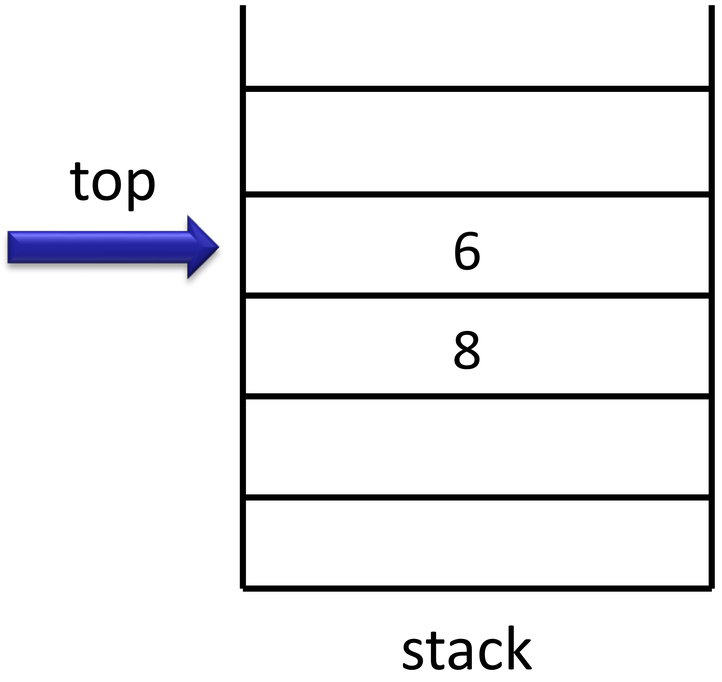


Operands are consumed from stack and put back onto stack



Stack Machine Simulator

```
var code : Array[Instruction]
var pc : Int   // program counter
var local : Array[Int]  // for local variables
var operand : Array[Int]  // operand stack
var top : Int
```




```
while (true) step

 def step = code(pc) match {
   case Iadd() =>
     operand(top - 1) = operand(top - 1) +   operand(top)
     top = top - 1  // two consumed, one produced
   case Imul() =>
     operand(top - 1) = operand(top - 1) * operand(top)
     top = top - 1  // two consumed, one produced
```

Stack Machine Simulator: Moving Data

```
case iconst(c) =>
  operand(top + 1) = c   // put given constant 'c' onto stack
  top = top + 1
case Igetlocal(n) =>
  operand(top + 1) = local(n)  // from memory onto stack
  top = top + 1
case Isetlocal(n) =>
  local(n) = operand(top)  // from stack into memory
  top = top - 1  // consumed
}
if (notJump(code(n)))
  pc = pc + 1   // by default go to next instructions
```

WebAssembly reference interpreter in ocaml:

https://github.com/WebAssembly/spec/tree/master/interpreter